

# Making a large treebank searchable online. The SoNaR case.

Vincent Vandeghinste, Liesbeth Augustinus

Centre for Computational Linguistics, KU Leuven  
Leuven

Email: [vincent@ccl.kuleuven.be](mailto:vincent@ccl.kuleuven.be), [liesbeth@ccl.kuleuven.be](mailto:liesbeth@ccl.kuleuven.be)

## Abstract

We describe our efforts to scale up a syntactic search engine from a 1 million word treebank of written Dutch text to a treebank of 500 million words, without increasing the query time by a factor of 500. This is not a trivial task. We have adapted the architecture of the database in order to allow querying the syntactic annotation layer of the SoNaR corpus in reasonable time. We reduce the search space by splitting the data in many small databases, which each link similar syntactic patterns with sentence identifiers. By knowing on which databases we have to apply the XPath query we aim to reduce the query times.

**Keywords:** corpus indexing, treebanks, search engine

## 1. Introduction

The research described in this work mainly concerns how to scale up syntactic search from a 1 million word treebank to a 500 million word treebank. In the *Nederbooms* project Augustinus et al. (2012) made the LASSY Small treebank (van Noord et al., 2013) available for online querying using GrETEL.<sup>1</sup> LASSY Small consists of 1 million words which have been syntactically annotated using the Alpino parser (van Noord, 2006), and which have been manually corrected.

The SoNaR corpus (Oostdijk et al., 2013) is a balanced corpus of written Dutch that consists of 500 million words, which are fully automatically tokenized, POS-tagged, lemmatized, and syntactically analysed (van Noord et al., 2013), using the Alpino parser.

Scaling up the treebank from a 1 million word corpus to a 500 million word corpus is not a trivial task. This paper describes the new architecture we had to adopt in order to allow querying the syntactic annotation of the SoNaR corpus in reasonable time.

In section 2 we describe some related work as well as our motivation to use XPath as a query language for the LASSY and SoNaR treebanks. In section 3 we describe the GrETEL 1.0 approach which we use for querying small treebanks. In section 4, we describe scaling up the GrETEL database architecture for very large treebanks. In section 5, the GrETEL 2.0 search engine is presented. Section 6 concludes and describes future work

## 2. Treebank Querying

Currently there exist many linguistic treebanks and almost as many query languages and treebanking tools to explore those treebanks.

The Penn Treebank (Marcus et al., 1993) should be queried with TGrep2 (Rohde, 2005) via a command-line interface,

the Prague Dependency Treebank (Hajič et al., 2006) can be queried through Netgraph (Mírovský, 2008), a client-server application. Several treebanking tools support (extensions of) the TIGER query language (König and Lezius, 2003), such as the standalone tool TIGERSearch (Lezius, 2002), or the online query engines TüNDRA (Martens, 2012; 2013) and INESS-Search (Meurer, 2012). For a comparison of some existing query languages, see Lai and Bird (2004).

In both LASSY Small and SoNaR the syntactic structures are represented in XML format, which can be visualised using the appropriate style sheet.<sup>2</sup> Each XML tree structure is isomorphous to the dependency tree of the syntactic annotation layer. This is not the case for XML representations of syntax trees in Tiger-XML format (König et al., 2003) or FoLiA format (Van Gompel and Reynaert, 2014), in which trees are represented as sets of nodes and links between those nodes. The use of isomorphous XML allows to query for syntactic structures using W3C standard tools such as XPath<sup>3</sup> and XQuery,<sup>4</sup> as explained in van Noord et al. (2013).

DTSearch (Bouma and Kloosterman 2002; 2007) and DACT<sup>5</sup> are two standalone tools for querying syntactic trees in Alpino-XML format using XPath. Since XPath is a standard query language and is already supported in the standalone tools for querying Dutch treebanks, we use it as a query language in GrETEL as well, as explained in more detail in section 3.

1 Greedy Extraction of Trees for Empirical Linguistics, <http://nederbooms.ccl.kuleuven.be/eng/gretel>

2 You can try this at <http://www.let.rug.nl/vannoord/bin/alpino> or <http://nederbooms.ccl.kuleuven.be/eng/alpino>.

3 <http://www.w3.org/TR/xpath/>

4 <http://www.w3.org/TR/xquery/>

5 <http://rug-compling.github.io/dact/>

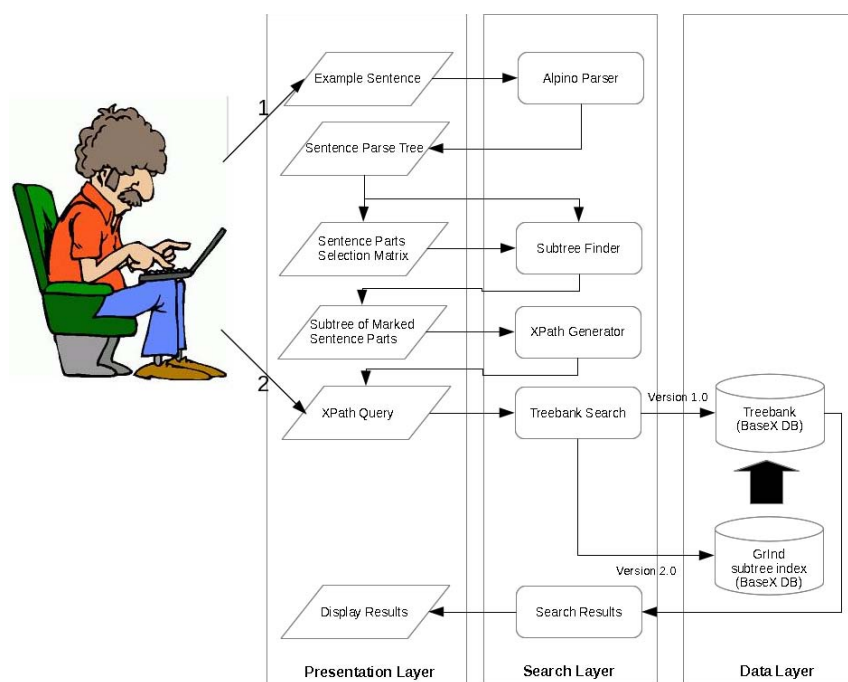


Figure 1: GrETEL architecture

This avoids the common complaint of users having to learn yet another query language. Van Noord et al. (2013) show how the queries presented in Lai and Bird (2004) can be *translated* to the Dutch grammar and fairly easily *converted* into XPath.

In order to support (non-technical) users that are reluctant towards learning any query language at all, we also implemented example-based search, a query system that does not ask for any formal input query. Tools related to that approach are the Linguist's Search Engine (Resnik and Elkins, 2005), another example-based query engine (which is no longer available) and the TIGER Corpus Navigator (Hellman et al., 2010), a SemanticWeb system that classifies and retrieves corpus sentences based on abstract linguistic concepts.

The system we present here is an online system, which shares the advantages of tools like TüNDRA and INESS-Search: They are platform independent and no local installation of the treebanks is needed. This is especially attractive for (very) large parsed corpora which require a lot of disk space.

### 3. GrETEL 1.0 (Lassy small)

Figure 1 presents the architecture of the GrETEL search engine. The user has two ways of entering a syntactic query.

The first approach, indicated by (1) in Figure 1, is called *Example-based Querying* (Augustinus et al., 2012; 2013)

which consists of a query procedure in several steps. The user provides an example sentence, containing the syntactic construction (s)he is querying. The input sentence is automatically syntactically analysed using the Alpino parser. In the *Sentence Parts Selection Matrix*, the user indicates which part of the entered example (s)he is actually looking for. The *Subtree Finder* extracts the query tree from the full parse tree. The *XPath Generator* automatically converts the query tree into an XPath expression. The XPath query is then used to search the treebank stored in the BaseX database system (Holupirek and Scholl, 2008), which is a native XML database system optimised for XQuery and XPath performance.<sup>6</sup> For example, if one is looking for constructions containing the adverbial phrase *lang niet altijd* “not always”, a possible input example is the construction *Het is lang niet altijd gemakkelijk* “It is far from easy”. After indicating the relevant parts of the input construction, GrETEL extracts the subtree in Figure 2, and turns it into the XPath expression in Figure 3.

<sup>6</sup> <http://basex.org/>

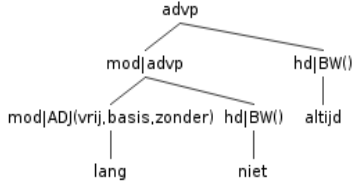


Figure 2: An example bottom-up subtree.

```
//node[@cat="advp" and node[@rel="mod"
and @cat="advp" and node[@rel="mod" and
@pt="bw" and @lemma="lang"]] and
node[@rel="hd" and @pt="bw" and
@lemma="niet"]] and node[@rel="hd" and
@pt="bw" and @lemma="altijd"]]
```

Figure 3: XPath based on Figure 2

Example-based querying has the advantage that the user does not need to be familiar with XPath, nor with the exact syntactic sugar of the XML in which the trees are represented, nor with the exact grammar implementation that is used by the parser or the annotators. Nevertheless, XPath querying greatly enhances the query flexibility compared to the example-based approach.

Therefore, the second approach, indicated by (2) in Figure 1, consists of directly formulating an XPath query describing the syntactic pattern the user is looking for. This query is then processed in the same way as the automatically generated query in the first approach. The online query engine is fast, but if one is looking for rare constructions, little or no results are found since the size of the treebank is rather small. We want to overcome this problem by including the large SoNaR corpus, but this introduces another challenge: Making such a large treebank searchable in *reasonable*-time, in order to implement it for online querying.

#### 4. GrInding the data per breadth-first pattern

The general idea behind our approach is to restrict the search space by splitting up the data in many small databases, allowing for faster retrieval of syntactic structures. We organise the data in databases that contain all bottom-up subtrees for which the two top levels (i.e. the root and its children) adhere to the same syntactic pattern. When querying the database for certain syntactic constructions, we know on which databases we have to apply the XPath query which would otherwise have to be applied on the whole data set. We have called this method GrETEL Indexing (GrInd).

As an example, take the parse tree presented in Figure 4.

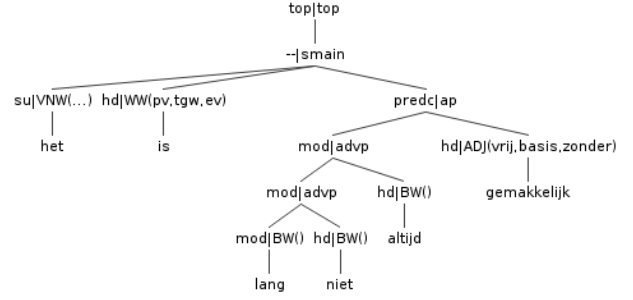


Figure 4: An example parse tree

In a top-down fashion, we take for each node all the bottom-up subtrees, i.e. all the subtrees that have only lexical elements as their terminals.<sup>7</sup> For instance, taking the example parse tree from Figure 4, for the --|smain node, we extracted the subtree with three daughters shown in Figure 5a, with two daughters in Figure 5b and with only one daughter in Figure 5c.

This procedure is applied recursively for each node in the parse tree. For every extracted subtree, we convert the children of the root into a string-based breadth-first pattern, inspired by Chi et al. (2005) and taken over from Vandeghinste and Martens (2010).<sup>8</sup> As the trees are dependency trees, the order of children is not important. Therefore, the children in the breadth-first representation are sorted in a canonical alphabetical order. Note also that the POS tags included in the trees contain more detailed information such as gender, number and agreement. In the breadth-first strings, only the general POS tag is used. For the subtrees in Figure 5, this amounts to the following breadth-first patterns respectively:

```
hd%ww_predc%ap_su%vnw
hd%ww_su%vnw
su%vnw_predc%ap
predc%ap_hd%ww
su%vnw
hd%ww
predc%ap
```

Those breadth-first patterns are combined with the category label of their root (in this case `smain`). To the file with that name we copy the XML of the subtree, adding the sentence identifiers indicating where these subtrees come from. We organise these breadth-first files per corpus component.<sup>9</sup>

<sup>7</sup> This definition differs from the definition of bottom-up subtrees used in Vandeghinste et al. (2013) in the sense that in this case, they do not have to be horizontally complete.

<sup>8</sup> Opposed to Vandeghinste and Martens (2010) only the children of the root are converted into breadth-first patterns.

<sup>9</sup> SoNaR contains 25 components, each containing a different text genre.

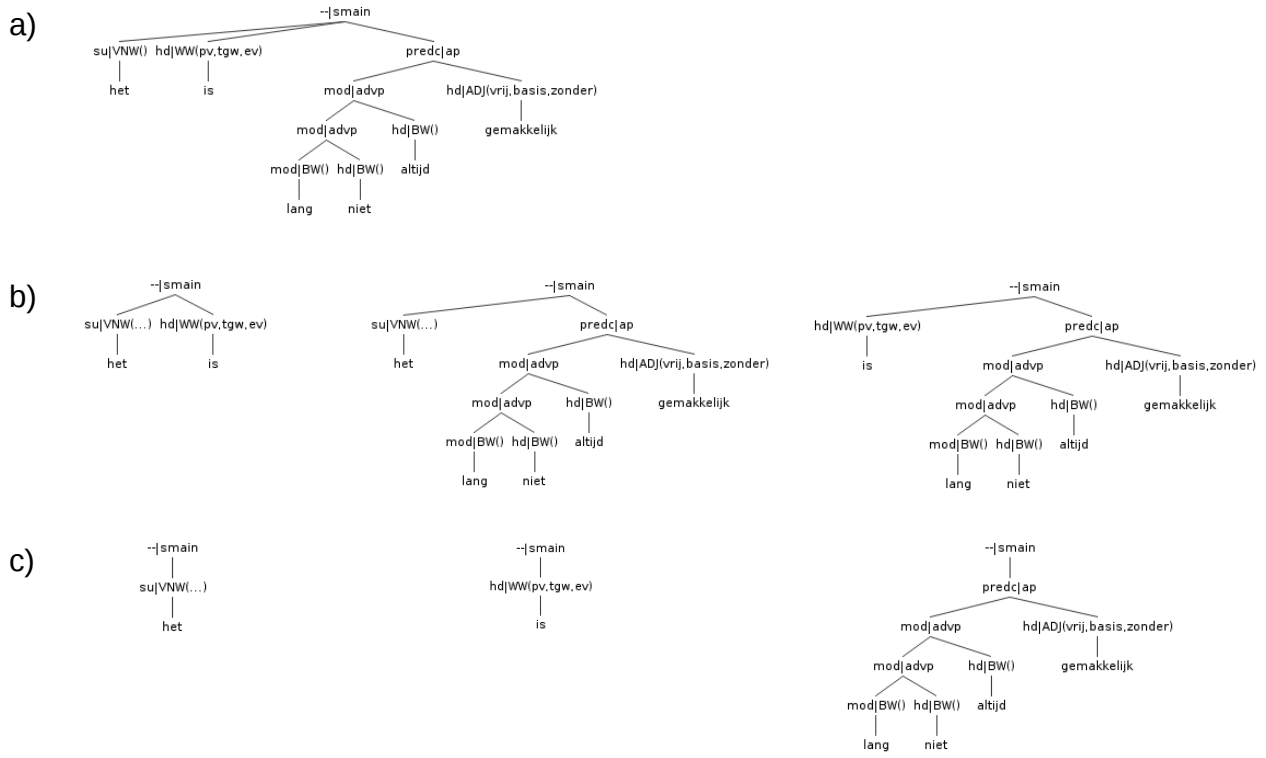


Figure 5: Bottom-up subtrees of the *smain* node in Figure 4

For example, in the WR-P-E-F component<sup>10</sup> of SoNaR, we have extracted all adverbial phrases that have as children an adverbial phrase (advp) as a modifier (mod) and an adverb (bw) as a head (hd), and put them in the following XML file `WRPEFadvphd%bw_mod%advp.xml`. The XML structure in Figure 6 corresponds to the subtree in Figure 2.

```

<treebank component="WRPEF" cat="advp" file="hd
%bw_mod%advp">
  <tree id="WR-P-E-F-0000000769.p.4.s.7" >
    <node begin="3" cat="advp" end="6" id="6"
    rel="mod">
      <node begin="3" cat="advp" end="5" id="7"
      rel="mod">
        <node begin="3" buiging="zonder" end="4"
        frame="adverb" graad="basis" id="8" lcat="advp"
        lemma="lang" pos="adv" positie="vrij"
        postag="ADJ(vrij,basis,zonder)" pt="adj"
        rel="mod" root="lang" sense="lang" word="lang"/>
        <node begin="4" end="5" frame="adverb" id="9"
        lcat="advp" lemma="niet" pos="adv" postag="BW()"
        pt="bw" rel="hd" root="niet" sense="niet"
        word="niet"/>
      </node>
      <node begin="5" end="6" frame="adverb" id="10"
      lcat="advp" lemma="altijd" pos="adv"
      postag="BW()" pt="bw" rel="hd" root="altijd"
      sense="altijd" word="altijd"/>
    </node>
  </tree>
</treebank>

```

Figure 6: XML containing a subtree

By extracting all the bottom-up subtrees from a given parse tree, and copying them to the files with the appropriate breadth-first filename, higher-level subtrees contain copies of lower level subtrees, and the size of the data grows considerably. In order to avoid copying the information from horizontally complete subtree patterns to the horizontally incomplete variants, we use `<include>` tags, indicating in which other files the queried pattern might occur as well. More general patterns are included in more specific patterns.

For instance, when looking for adverbial phrases (advp) that have a modifying adverbial phrase (mod|advp) as daughter, one should also look for this pattern in the file that contains the adverbial phrases that have a modifying adverbial phrase AND a head adverb (hd|bw) as daughters, resulting in a file `WRPEF/advp/bfmod%advp.xml` with the following data:

```

<treebank component="WRPEF" cat="advp" file="mod
%advp">
  <include file="WRPEFadvpmod%advp_hd%bw" />
</treebank>

```

Figure 7: XML of a subtree that is included in another database

When a subtree is found that matches the XPath query, we retrieve the sentence identifier, as indicated by the `id` feature in the `<tree>` tag, as shown in Figure 6, and

<sup>10</sup>The WR-P-E-F component contains press releases.

display the full sentence and optionally the full parse tree as retrieved from the original treebank.

## 5. Querying the data (GrETEL 2.0)

Similar to the query engine for Lassy small, we use BaseX (Holupírek and Scholl, 2008), a native XML database system, as an XPath query engine for SoNaR. We have created over 10 million databases in BaseX, each with a name that is indicative of the components and patterns that are described therein. GrInd allows us to query SoNaR in a faster and more efficient way, whereas earlier attempts to query the corpus resulted in memory issues.

However, some queries still take too long for online searching, which caused a browser time-out. Therefore, the current system outputs a sample of the results and their frequency during the first  $n$  seconds of querying.<sup>11</sup> By caching the queries and the results we avoid querying the same construction multiple times, and it allows us to return the final results immediately.

In the next version we plan to implement a messaging system which notifies when the user can find the results once the search action is completed.

As described in section 3 we have two ways of feeding queries to the syntactic search engine: Through an example and through an XPath expression. The first version of the online search engine for SoNaR only allows the *example-based querying* method.

When we are querying via an example (query method 1 in Figure 1), the search engine extracts a bottom-up query subtree from the parse tree of the (natural language) input example. When we are querying the treebank looking for this bottom-up subtree, we know in which BaseX database we have to look, as we can construct the name of the database based on the query subtree. As described in the previous section, the database name depends on the syntactic category of the root and on the syntactic categories and dependency relations of the children of the root. It is only in trees in this database that the query subtree can occur, and therefore we do not need to look in the rest of the treebank. The query subtree is also automatically converted into XPath, and it is this XPath expression that is used as a query in the relevant databases. When we are querying with an XPath expression (query method 2 in Figure 1), there is no subtree that can be used to retrieve the relevant databases. Instead, we have to extract the largest bottom-up subtree that complies with the XPath expression in order to determine where to look for the relevant patterns. This is not a trivial process, as XPath can contain conjunctions (AND), optionality (OR), and negation (NOT), defining multiple query trees in one query, not necessarily with the same root category label and the same children of the root, and hence not necessarily in the same database. Solving this issue remains future work.

<sup>11</sup> Currently,  $n$  is set to 60 seconds, but parametrisable.

## 6. Conclusions and future work

We have described how we have organised the data of a very large treebank of written Dutch in order to search syntactic constructions within reasonable query times. When the user is interested in frequencies of syntactic patterns, query times will be longer, as all the relevant databases from all the relevant components need to be queried. Nevertheless we are not aware of any other approach towards large treebank querying that allows faster querying.

The work described in this paper is still in progress. We still have to implement extracting the largest bottom-up tree complying with an XPath expression, which is a non-trivial task, as XPath expressions can contain optionality and negation. As already mentioned, we will include a messaging system to work around excessive query times, which we still occasionally expect. Finally, we also plan to apply this architecture to treebanks for Afrikaans and English, increasing the coverage of the GrETEL search engine.

## 7. Acknowledgements

The research presented in this paper is part of GrETEL 2.0, a project on the adaption of the GrETEL search engine for querying very large treebanks, sponsored by CLARIN-NTU.

## 8. References

- Liesbeth Augustinus, Vincent Vandeghinste, Ineke Schuurman, and Frank Van Eynde (2013). Example-Based Treebank Querying with GrETEL – now also for Spoken Dutch. In *Proceedings of the 19th Nordic Conference on Computational Linguistics (NoDaLiDa 2013)*. NEALT Proceedings Series 16. Oslo, Norway. pp. 423-428.
- Liesbeth Augustinus, Vincent Vandeghinste, and Frank Van Eynde (2012). Example-Based Treebank Querying. In *Proceedings of LREC 2012*. Istanbul, Turkey. pp. 3161-3167
- Gosse Bouma and Geert Kloosterman (2002). Querying Dependency Treebanks in XML. In *Proceedings of LREC'02*. Las Palmas, Spain. pp. 1686-1691.
- Gosse Bouma and Geert Kloosterman (2007). Mining Syntactically Annotated Corpora with XQuery. In *Proceedings of the Linguistic Annotation Workshop*. Prague, Czech Republic. pp. 17-24.
- Yun Chi, Siegfried Nijssen, Richard Muntz, and Joost Kok (2005). Frequent Subtree Mining An Overview. *Fundamental Informatics, Special Issue on Graph and Tree Mining*. pp. 1001-1038.
- Jan Hajič, Jarmila Panevová, Eva Hajičová, Petr Sgall, Petr Pajas, Jan Štěpánek, Jiří Havelka, Marie Mikulová, Zdeněk Žabokrtský and Magda Ševčíková-Razimová

- (2006). Prague Dependency Treebank 2.0. CD-ROM LDC2006T01, LDC, Philadelphia.
- Sebastian Hellmann, Jörg Unbehauen, Christian Chiarcos, and Axel-Cyrille Ngonga Ngomo (2010). The TIGER Corpus Navigator. In *Proceedings of the The Ninth Workshop on Treebanks and Linguistic Theories (TLT9)*. Tartu, Estonia. pp. 91-102.
- Alexander Holupirek and Marc H. Scholl (2008). An XML Database as Filesystem in Userspace. In *Proceedings of the 20. GI Workshop on Foundations of Databases, August 2008*. School of Information Technology, Germany. pp. 31-35.
- Ester König and Wolfgang Lezius (2003). The TIGER language - A Description Language for Syntax Graphs, Formal Definition. Technical report. IMS, University of Stuttgart, Germany.
- Esther König, Wolfgang Lezius, and Holger Voormann (2003). TIGERSearch User's Manual. IMS, University of Stuttgart, Germany.
- Catherine Lai and Steven Bird (2004). Querying and updating treebanks: a critical survey and requirements analysis. In *Proceedings of the Australasian Language Technology Workshop*. Sydney, Australia. pp. 139-146.
- Wolfgang Lezius (2002). TIGERSearch: ein Suchwerkzeug für Baumbanken. In *Proceedings of KONVENS-02*. Saarbrücken, Germany.
- Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz (1993). Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313-330.
- Scott Martens (2012). TüNDRA: TIGERSearch-style treebank querying as an XQuery-based web service. In *Proceedings of the joint CLARIN-D/DARIAH Workshop "Service-oriented Architectures (SOAs) for the Humanities: Solutions and Impacts" (DH 2012)*. Hamburg. pp. 41-50.
- Scott Martens (2013). TüNDRA: A Web Application for Treebank Search and Visualization. In *Proceedings of the The Twelfth Workshop on Treebanks and Linguistic Theories (TLT12)*. Sofia, Bulgaria. pp. 133-143.
- Paul Meurer (2012). INESS-Search: A search system for LFG (and other) treebanks. In *Proceedings of the LFG'12 Conference*. LFG Online Proceedings. Stanford, CSLI Publications. pp. 404-421
- Jiří Mirovský (2008). Netgraph Query Language for the Prague Dependency Treebank 2.0. *The Prague Bulletin of Mathematical Linguistics* No. 90. pp. 5-32.
- Nelleke Oostdijk, Martin Reynaert, Véronique Hoste, and Ineke Schuurman (2013). The Construction of a 500-million-word Reference Corpus of Contemporary Written Dutch. In Peter Spyns and Jan Odijk (eds.): *Essential Speech and Language Technology for Dutch: resources, tools and applications*. Springer.
- Philip Resnik and Aaron Elkiss (2005). The Linguist's Search Engine: An Overview. In *Proceedings of the ACL Interactive Poster and Demonstration Sessions*. Ann Arbor. pp. 33-36.
- Douglas L.T. Rohde (2005). TGrep2 User Manual.
- Vincent Vandeghinste, Scott Martens, Gideon Kotzé, Jörg Tiedemann, Joachim Van Den Bogaert, Koen De Smet, Frank Van Eynde, and Gertjan van Noord (2013). Parse and Corpus-based Machine Translation. In Peter Spyns & Jan Odijk (eds.): *Essential Speech and Language Technology for Dutch: resources, tools and applications*. Springer.
- Vincent Vandeghinste and Scott Martens (2010). Bottom-up transfer in Example-based Machine Translation. In François Iyon and Viggo Hansen (eds.) In *Proceedings of the 14th International Conference of the European Association for Machine Translation (EAMT-2010)*. Saint-Raphael, France.
- Maarten van Gompel and Martin Reynaert (2014). FoLiA: A practical XML format for linguistic annotation - a descriptive and comparative study. *Computational Linguistics in the Netherlands Journal* 3:63-81.
- Gertjan van Noord. 2006. At Last Parsing Is Now Operational. In *TALN 2006*. pp. 20-42.
- Gertjan van Noord, Gosse Bouma, Frank Van Eynde, Daniel de Kok, Jelmer van der Linde, Ineke Schuurman, Erik Tjong Kim Sang, and Vincent Vandeghinste (2013). Large Scale Syntactic Annotation of Written Dutch: Lassy. In: Peter Spyns and Jan Odijk (eds.): *Essential Speech and Language Technology for Dutch: Resources, Tools and Applications*. Springer.